Tack: Learning Towards Contextual and Ephemeral Indoor Localization with Crowdsourcing

Liyao Xiang, *Student Member, IEEE,* Tzu-Yin Tai, Baochun Li, *Fellow, IEEE,* and Bo Li, *Fellow, IEEE*

***Abstract*—At events such as conferences, indoor localization are both contextual and ephemeral, in that localization is only needed within the context of and for the duration of the event. As such, the costs and requirements of providing such services need to be minimal. In this paper, we design, implement, and evaluate *Tack*, a new mobile application framework that is specifically engineered to support such contextual and ephemeral indoor localization during an event. To provide location-based services with Tack, an event organizer only needs to bring and place a small number of (reusable) beacons around the venue before the event begins. As a system framework, Tack uses a combination of known beacon locations, contacts over Bluetooth Low Energy, crowdsourcing, and dead-reckoning to estimate and refine user locations. To make our location estimates more accurate, we embrace the inherent nature of beacons, design crowdsourcing-based inference algorithms, and present an extensive evaluation by running real-world experiments with iOS devices and beacons. *Tack* has been implemented as an open-source framework on the iOS platform, and can be used by mobile applications designed for events with location-based services.**

***Index Terms*—Smart devices, Bluetooth, Crowdsourcing, Indoor environments, Localization, Mobile Computing, Inference Mechanisms, Particle filters**

I. Introduction

There is often a need to provide location-based services within the context of a large indoor event at a venue without any infrastructure support for indoor localization. At a large conference, for example, estimates of user locations are often beneficial for spontaneous social interaction, in situ headcounts in conference rooms, as well as location-specific push notifications. Such needs for indoor localization are both *contextual* and *ephemeral*, in that location-based services are only needed within the context of and for the duration of the event, rather than permanently. As well, the incurred costs for the required infrastructure, if any, will be borne by the event organizers.

While there exists a large volume of previous work in the area of indoor localization, we found surprisingly few that would be conceptually satisfactory for such contextual and ephemeral indoor localization during an event. On one end of the spectrum, some indoor localization schemes from both academia and the industry have achieved a high degree of accuracy, by relying on either specialized hardware such as antenna arrays [17], or extensive offline measurements of wireless signal fingerprints [3]. It is certainly not realistic to expect event attendees to own hardware more specialized than off-the-shelf smartphones. The needs for fingerprinting or war-driving (*e.g.,* [3], [18]), on the other hand, are inherently *venue-specific*, rather than *event-specific*. The labour and material costs for such fingerprint measurements or war-driving are too overwhelming for contextual and ephemeral indoor localization.

On the other end of the spectrum, there has been a recent trend in the literature to mitigate, or even completely avoid, the costs incurred by fingerprinting or war-driving the venue [5], [6], [13], [15], [20]. These existing works focused on mobile devices, and used a combination of noisy dead-reckoning and various types of calibrating signals, such as fixed beacons [5], encounters [8], signal-based landmarks [13], or camera-based [20] navigation. While these strategies offered inspiring ideas, they were not immediately applicable in mobile applications that require contextual and ephemeral indoor localization, due to their additional assumptions on the venue (*e.g.*, elevators) or the user (*e.g.* permissions to use the camera).

In contextual and ephemeral localization, what do these event organizers and their mobile applications need, anyway? First, the costs for any infrastructure that needs to be established by the event organizer before the event will need to be *minimal*. Second, the energy costs to the attendees with their smartphones have to be negligible, or else permissions will not be granted. In this work, our fundamental objective is to achieve the best possible localization accuracy within these practical constraints.

In this paper, we design, implement, and evaluate *Tack*, a new mobile application framework that is specifically engineered to support such contextual and ephemeral indoor localization during an event. When designing Tack, we share the pessimism with existing work that pure dead-reckoning using smartphone sensors is inherently noisy, and should only be supplementary. To provide an initial infusion of accurate location data, we recognize that Bluetooth Low energy (BLE) has overwhelming advantages: it is energy-efficient; ubiquitous in that it is supported by all modern smartphones; and above all, used by the iBeacon-compatible transmitters [1] that can be inexpensively acquired (less than $10 each), reusable, and easily tacked out of sight at any indoor location. With Tack, we take advantage of a collection of iBeacon-compatible transmitters, referred to simply as *beacons* in this paper, to provide accurate location data. To provide location-based services with Tack, an event organizer only needs to bring and place a small number of (reusable) beacons around the

L. Xiang, T. Tai, B. Li are with the Department of Electrical and Computer Engineering, University of Toronto, Toronto, ON M5S 3G4, Canada (e-mail: liyaox@ece.utoronto.ca, nina.tai@mail.utoronto.ca, bli@ece.toronto.edu). B. Li is with the Department of Computer Science and Engineering, Hong Kong University of Science and Technology. The research was supported in part by grants from RGC under the contracts 615613, 16211715 and C7036-15G (CRF), a grant from NSF (China) under the contract U1301253.

Manuscript received September 22, 2016; revised January 13, 2017; accepted January 26, 2017.

venue before the event begins.

A unique advantage of a large indoor event is the density of its attendees within a reasonably confined space, such as a conference center or a hotel. Tack uses such density to its advantage: thanks to the recent operating system support for any smartphone to act as both a BLE *advertiser* and *scanner* at the same time, Tack refines user location estimates using crowdsourcing as they encounter one another. In the *advertiser* mode, a device broadcasts its virtual beacon ID to nearby BLE devices; while in the *scanner* mode, the device listens to advertisements. Operating in both modes allows a pair of devices to detect their ranges from each other, which is the foundation for our crowdsourcing algorithm.

While promising, our design needs to be improved to address new challenges that are unique to BLE beacons and mobile devices. Due to its lower energy costs, the operating range of BLE is only several meters, and distance estimates using signal strengths are inherently noisy. Intuitively, the number of beacons, each with known locations, will also affect the accuracy of location estimates in a substantial way. The critical challenge is how crowdsourcing, beacons, and dead-reckoning can work together harmoniously towards improving localization accuracy.

To overcome the lack of accuracy of distance estimates with BLE beacons, we propose to formulate a maximum *a posteriori* (MAP) inference problem based on graphical models to seamlessly integrate crowdsourcing, beacons, and dead-reckoning. We present models that are progressively more complicated as more empirical observations are incorporated, with improved localization accuracy. Using real-world tests on iOS devices and beacons, we have extensively evaluated Tack with respect to its accuracy. In general, Tack has achieved an accuracy of 2 to 4 metres with moderate energy costs and latency in our experiments. Overall, Tack only relies on a few beacons and does not require the support of external power sources, thus is very easy and cost-efficient to deploy. Tack has been implemented as an open-source framework on the iOS platform, and can be readily used by mobile applications designed for events with location-based services.

II. Related Work

The literature on indoor localization is vast, but the closely related work to this paper falls into the following categories.

Fingerprinting-based localization. This line of work utilizes the unique set of WiFi [7], [12], [18] or magnetic [16] fingerprints to evaluate user locations. For WiFi fingerprinting, a serious problem is that the user may unintentionally leak its WiFi SSID, which threatens its privacy. Another issue with such a system is that it assumes multiple access points are present, which may not be true in reality. Magnetic fingerprint, on the other hand, may not be unique in a large indoor space and cannot withstand infrastructure changes. In our localization system, we rely on the estimated distance instead of any fingerprinting.

Range-based localization. In these schemes, absolute point-to-point distance or angle estimates are used for calculating positions. Depending on the characteristics of wireless signals, methods of obtaining ranges vary from Time of Arrival, Angle of Arrival, to Received Signal Strength Indicator (RSSI) values. Approaches include multilateration [3], probability inference, and wireless signal model based methods [4]. These localization systems mostly suffer from multi-path fading, background interference, and other irregular signal propagation characteristics over long distances and large spaces. In our localization framework, we address this drawback by using raw distance estimates only when the beacons are in close proximity to each other when the distance estimate is most likely to be accurate, and complement it with the technique of dead reckoning, contacts, and crowdsourcing. Moreover, we are among the first works that adopt the BLE virtual beacon mode for each user's device to obtain user-to-user distance observations in a localization framework.

Dead reckoning with landmarks. Traditionally, we compute the motion trajectory of a smartphone by using its accelerometer and compass. Due to the inherent noise in mobile sensors, the trajectories are often accurate at the beginning, and gradually drift away. *Landmarks* with known location are usually provided to reposition the user from time to time to cancel out the cumulative error, in typical ways as [2], [11], [13] do. The accuracy of localization is naturally higher as the density of deployed repositioning anchors is higher. [2], [13] proposed to use sensor signatures that are unique in the WiFi-subspace as landmarks; however, this kind of landmarks is not consistent across different smartphone platforms. WiFi-Marks [11] are special locations where the received WiFi signal strength changes from increasing to decreasing, representing the nearest position to a WiFi access point. But one never knows its actual position relative to the access point even at the tipping point, let alone the fact that such landmarks are few and occur opportunistically anyway.

In contrast, we combine nearby beacons with contacts over BLE to reposition users in our framework, greatly improving repositioning opportunities throughout time and space, and expanding the influence range of the repositioning effect to users beyond the one in contact. Such a positive effect increases with larger crowds roaming the venue. These virtual landmarks are not only device-independent, but also observed frequently.

Localization using contacts. Opportunistic user interactions are used to develop human escort services [5] or to improve indoor localization [8]. In previous works [8], [14], contact is used in a "macro" environment to select the path a user takes or the location it resides at; we inherit the method and improve it to integrate with particle filters. In previous implementations, contacts can be spotted by a real person, or detected by audio or wireless signals, which are subjective to environmental noise. Classic Bluetooth for contacts was claimed to be slow in discovering short-lived encounters (around 100 ms). BLE, on the other hand, significantly reduces the discovering latency (6 ms for the non-connection state). In Tack, contacts via BLE is not only faster, but also free — each user acts as a virtual beacon, broadcasting its own location estimates.

Crowdsourcing for indoor localization. In existing crowdsourcing schemes [9], [15], a central server is designed to collect wireless signal fingerprints, sensor signatures, or user

trajectories from different users to alleviate the effort of the offline training phase. One important problem of these schemes is that the localization service cannot be provided instantaneously: they typically need a "warm-up" phase for collecting some user trajectories to train a model of the wireless fingerprints or sensor signatures. Our crowdsourcing approach allows the localization system to work effectively as soon as some user data are collected.

III. Tack: Overview and Challenges

Overview. Existing works have all assumed a *dense* deployment of beacons, precisely because the accuracy of distance estimates is only acceptable when the devices to be localized are very close to the beacons. Yet, for our purpose of contextual and ephemeral localization with a large number of users, it would not be economical to deploy a high density of beacons across the venue of an event. However, if we place beacons *sparsely* in the venue, the fast-decaying BLE signals would make it almost impossible for all users to position themselves.

We extend the horizon using *dead reckoning* (DR). DR is a navigation process of calculating one's current position by advancing from a previously determined position based on the number of steps of a user, and each step's direction headings. In Tack, we designed a software *step counter* by filtering the accelerometer readings. Step counters are also supported by modern smartphone hardware, such as the M7/M8 processors on recent iOS devices. The heading direction can be easily obtained from the magnetometer reading on smartphones.

However, when dead reckoning is used independently without repositioning, the localization result can easily drift away due to accumulated error. Thus, we propose to combine DR with BLE beacons through *particle filters*: while DR tracks users' positions, the BLE signals observed from beacons or mobile devices are adopted to correct users' trajectories. Particle filters will be thoroughly introduced in Sec. IV-A.

To take advantage of crowdsourcing, we further introduced *virtual beacons* to enhance the density of beacons. In Tack, each fixed beacon is configured with its own geo coordinates on the given floor plan. At the very beginning, each mobile user pulls all beacons' coordinates from the server and turns on Bluetooth dual mode to broadcast as a virtual beacon, and to detect both fixed beacons and other mobile users in its vicinity. When running the service, all mobile users upload their data to the server and download other users' data from the server periodically. Our crowdsourcing algorithm takes all data as input, and outputs the position estimate of the user. Computation can be performed either at the server or locally on the user device. In our prototype system, we choose to compute locally and accelerate the computation by using the hardware-assisted `Accelerate` framework on the iOS platform.

Fig. 1 shows an architectural overview of our design. The key issue in Tack is how to combine different parts — dead reckoning, BLE signals, and virtual beacons all together to allow each mobile user to locate itself as accurately as possible while minimizing the deployment costs. We will introduce the challenges in each part respectively, and show how our design overcomes these drawbacks to make all parts work seamlessly together to deliver position estimates that are as accurate as possible.

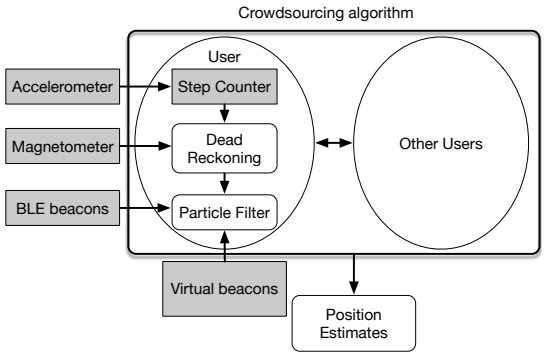


Fig. 1. Tack: Architectural overview.

Challenge 1: Noisy distance estimates. Even though the use of beacons for indoor localization has been explored by both industry (such as Estimote and Indoo.rs Inc.) and academia [19], it is a problem that remains elusive and far from solved. The primary reason is simple: due to the low energy consumption imposed by BLE, any distance estimates based on Received Signal Strength Indication (RSSI) are inherently noisy, and are thus not suitable for triangulation or similar algorithms that are designed for localization using multiple transmitters.

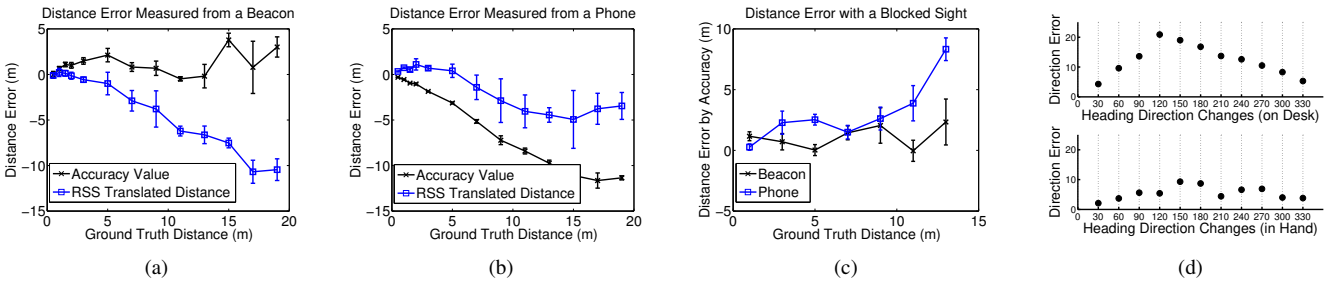
To illustrate these challenges quantitatively, we have conducted several experiments on Estimote beacons and iPhones to gain a better understanding of BLE and beacons. The *Core Location* framework on the iOS platform has conveniently provided an estimate of distance based on RSSI measurements, called `accuracy`, in the unit of meters from the beacon. Besides the estimate, the `CLBeacon` class also provides raw RSSI measurements called `rssi`. To find out which one is a better indicator of distance, we translate `rssi` into a measured distance D using the standard free-space radio propagation formula:

$$\text{RSSI}_m = \text{RSSI}_0 - 10n \log_{10} D + x_\sigma \tag{1}$$

Here, RSSI_m is the measured RSS on the user's phone, RSSI_0 is the RSS from the beacon at a distance of one meter, n is the rate at which RSS falls with distance depending on the local environment, and x_σ is a lognormal distributed random variable accounting for the slow-fading phenomenon. We adopt the recommended values for these parameters.

Our experiments are conducted in a hallway with no obstacles in sight and no other BLE devices turned on. In the two groups of experiments, we fix the position of a beacon or an iOS device, have the testing phone (iPhone 6S) placed at certain distances from it, and measure `accuracy` and `rssi`. We collect samples for multiple runs to compute the mean and standard deviation of the distance errors.

When measuring distances from a beacon, as Fig. 2(a) shows, the distance error obtained from the `accuracy` value is smaller with a modest standard deviation than that translated



(a) (b) (c) (d)

Fig. 2. Distance estimates using BLE (Sensors) are inherently noisy.

from `rssi`. On the contrary, when measuring distances from a phone, Fig. 2(b) shows that the translated distance using `rssi` enjoys a smaller error than using the `accuracy` value. This is because the `accuracy` value provided by the iOS framework is tuned for beacons rather than a smartphone; a smartphone normally has a higher transmission power than a beacon, and as a result, the distance it reports is usually less than the ground truth.

Despite the difference between the values reported by `accuracy` and `rssi`, we are also curious about the distance errors when there are obstacles in the environment. We further measured the distances reported by `accuracy` in the same hallway with the beacon (phone) blocked by obstacles. The result is depicted in Fig. 2(c). As we can tell, compared to the open space, the reported values are mostly above the ground truth, but are still close to them, particularly when the actual distance is less than 10 meters in between.

But regardless of the situation, the overall bad news is that we are not able to assume that distance estimates are accurate unless the actual distance is small. Moreover, in the presence of a beacon, the `accuracy` value is generally a better indicator of the distance estimate. This is because `accuracy` is tuned considering the surrounding environment, and is usually used to distinguish different objects in the same region of a beacon, according to Apple's document. Although quite noisy, the distance estimate is sufficiently accurate for our localization algorithm as we will discuss later.

Challenge 2: Noisy sensor readings. Besides noisy distance estimates, sensors used in DR pose another significant challenge. We have conducted new experiments to quantify the errors in the heading direction from magnetometer sensors on smartphones. Fig. 2(d) shows the error when turning the smartphone's heading from the true north. The true north is obtained by the compass and is recorded beforehand. As we can observe, the direction error reaches 20 degrees when the direction change is 120 to 180 degrees. Overall, the heading direction measured using a magnetometer has an error of around 5 degrees on average when the user holds it in her hand, and the error is slightly higher (around 10 degrees) when the phone is placed flat on a desk.

The step counter may also be another source of error. To quantitatively measure this source of error, we have conducted an experiment by asking 5 users to hold their phones and walk at different paces repeatedly. As shown in Fig. 3(a), a steady pace introduces the least amount of error, mainly because the filter parameter in the step counter is tuned according to a normal speed. On average, the error is only 1.5 steps for every 50 steps. In the worst case, the error is fewer than 3.5 steps. Such errors are not significant since indoor users usually take fewer than 50 steps before they are repositioned by contacts with beacons.

Our experiments so far have clearly shown that, with both noisy distance estimates and noisy sensor readings, new algorithms need to be designed to compute position estimates as accurately as possible, taking such noise into account.

IV. A Local View: A Probabilistic Approach

To introduce the localization inference system, we first take a single user's view. Each position is represented by a multi-dimensional variable. Some variables have more significant prior than others; for example, the fixed beacons with known positions can be represented as a Dirac delta distribution. Since the initial positions of the mobile users are unknown, their positions can be considered as uniform distributions over the floor plan. We use particles to represent each geo-distribution and propose augmented particle filters like the model in [9].

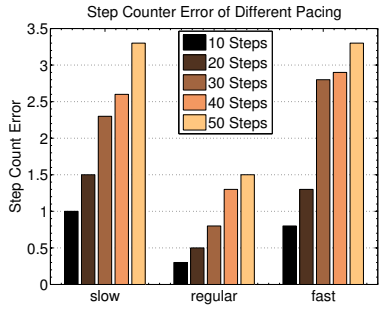
A. Augmented Particle Filters

In control theory, particle filters are used to improve the tracking accuracy of time-varying variables of interest, by constructing a sample-based representation of the targeted variables' probability density function (pdf). In particular, its performance exceeds other filtering methods, such as Kalman filters, in cases where variables are non-linear and non-Gaussian. In our localization system, as most of the previous work, we are interested in tracking the locations of users with their smartphones, represented by (x, y) coordinates in the floorplan of the venue.

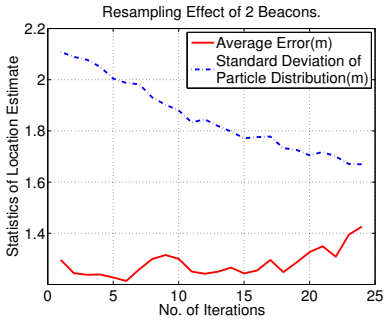
We begin our discussions from the principles of DR. We consider a set of particles $\mathbf{S} = (S_1, ..., S_N)$ as a discrete representation of the probability distribution of locations. Let $S_r^k = \{x_r^k, y_r^k, l_r^k, d_r^k, w_r^k\}$ denote the rth particle at the kth iteration. Here (x_r, y_r) jointly represents the geo-coordinate, l_r denotes the disturbance in stride length of each step, d_r is the disturbance in user's heading direction and w_r means the weight of the rth particle, suggesting how likely it represents the true user location.

Particle filtering implements a recursive Bayesian filter using the Sequential Monte-Carlo method. In the context of dead reckoning using smartphone sensors, the algorithm goes iteratively through the following four phases in each time slot to estimate user locations:

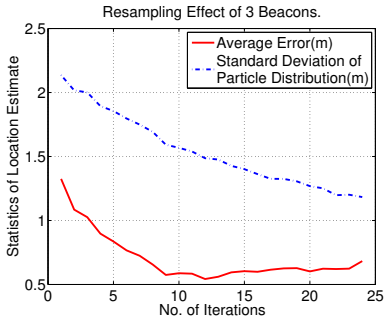
♢ *Initialization.* At time slot 0, each particle's position (x_r, y_r) is initialized to be uniformly distributed on the entire



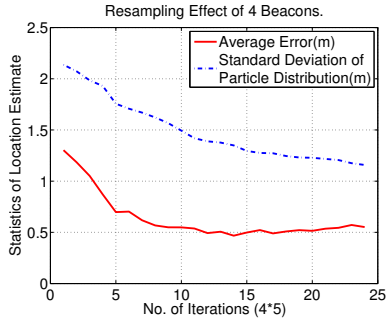
(a) Step counter error of different pacing.



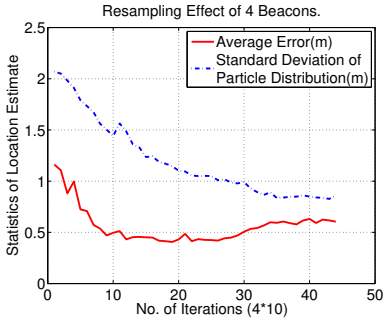
(b) 2 beacons, average $\sim$ 3m to each beacon, 4×5 iterations.



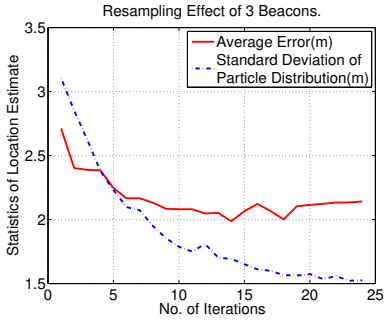
(c) 3 beacons, average $\sim$ 3m to each beacon, 4×5 iterations.



(d) 4 beacons, average $\sim$ 3m to each beacon, 4×5 iterations.



(e) Error and convergence with 4 beacons, 4×10 iterations.



(f) Error and convergence with 3 beacons, average $\sim$ 6m to each beacon.

Fig. 3. Position estimate error and convergence for each iteration in different settings (except (a) about step counter errors).

region with equivalent weight. If the starting location is given, the position is normally distributed around the starting location. From time slot 1 onwards, the initial position is the result from the last time slot.

♢ *Prediction.* As the user takes a step, we decompose the step into two orthogonal components: the distance traveled and the orientation of the step. The distance traveled is computed with $\beta + l_r$, where β is the average length of the user stride, and l_r denotes the disturbance in each step.

The orientation consists of the heading offset θ, the real-time magnetometer reading α^k, and its disturbance d_r. In practice the heading direction is obtained as follows. Before localization, we record the magnetometer reading when placing the device parallel to Y axis. This value is the heading offset that does not vary much throughout the same venue. During localization, we obtain α^k by the magnetometer reading. The sum (difference) of θ and α^k is the current heading direction from Y axis. Both l_r and d_r are drawn from the empirical distribution according to our previous test results in Sec. III. When a user takes a single step, all the variables in each particle are updated to its predicted state. After the kth step, the rth possible location (x_r^k, y_r^k) is predicted as:

$$\begin{pmatrix} x_r^k \\ y_r^k \end{pmatrix} = \begin{pmatrix} x_r^{k-1} + (\beta + l_r)\cos(\theta + \alpha^k + d_r) \\ y_r^{k-1} + (\beta + l_r)\sin(\theta + \alpha^k + d_r) \end{pmatrix} \quad (2)$$

♢ *Update.* The weight of each particle w_r^k is now recalculated as the likelihood that the particle S_r^k represents the actual location of the user:

$$w_r^k = p(S_r^k). \quad (3)$$

In traditional robotic localization, w_r^k is computed using a joint Gaussian error model for the sensor data [10]; in [9], particles violating map constraints are assigned a weight of zero. In our system, the weight of each particle is determined by the probability distribution of the user location, which is conditional upon the beacons' locations and the measured distances in between. We will later explain how to update the particle weight.

♢ *Resampling.* With the weight of each particle updated, some particles have drifted far enough that their weights are too small to contribute to the probability distribution of the user location. The particle population is then resampled by eliminating the ones with small weights and duplicating the ones with higher weights. Mere duplication will lead to the depletion of particles, so we actually generate a new particle drawn from a random distribution centered around the chosen particle. After repeating this step several iterations, most of the particles should be converged to an area. The user's position at the current time slot can be estimated by averaging over its particle set.

B. Location Probabilities

From the viewpoint of a user, its position distribution $p(S_r^k)$ is computed as the product of distance errors from any observed users and beacons. Intuitively, the smaller the distance error, the more probable that a position represents the ground truth.

There may be one or multiple fixed beacons or mobile users in the proximity of a user. The geo-locations of the fixed beacons are known *a priori*, and we assume the positions of nearby mobile users have already been computed recursively. Suppose that a nearby beacon or the reference user with a known position is located at (x_j, y_j), and we probabilistically

infer the unknown user's location by using *distance estimates*, denoted as $\bar{D}_j$, from the reference points. For each potential location (x_r, y_r), let its distance to the jth reference point be D_{rj}, and the distance estimate error is $\bar{D}_j - D_{rj}$.

We assume that the error in distance estimates is normally distributed with zero mean, which is a reasonable assumption according to the experimental results in Sec. III. With respect to the jth reference point, the probability that the location (x_r, y_r) represents the true location is:

$$p_j(x_r, y_r) = \frac{1}{\sqrt{2\pi}\sigma} \exp(-\frac{(\bar{D}_j - D_{rj})^2}{2\sigma^2}), \tag{4}$$

When multiple independent reference points exist, the probability $p(x_r, y_r)$ for the location (x_r, y_r) to be the true location is:

$$p(S_r) = \prod_{j \in \text{ref}} p_j(x_r, y_r), \ \forall r \in \{1, ..., N\}. \tag{5}$$

Whenever a user finds at least one nearby reference point, the weights of all its particles can then be computed during the update phase by using Eqn. (3) and (5). Then the potential locations for this user can quickly be narrowed down.

C. Resampling with Beacons

In the implementation of our augmented particle filters, we are curious about the convergence criteria and how they affect the resulting accuracy. Thus we run a series of small-scale experiments to verify our assumptions.

In a typical office environment, we record the particles of a user per iteration under the influence of nearby beacons. We vary the number of beacons and the average distances to them to figure out how these factors affect convergence and the resulting accuracy.

Fig. 3(b)-3(d) show the resampling effect of the user when it has an average of 3m to each beacon, with 2, 3 and 4 beacons in presence respectively. Because the scanning interval is set to be $\sim$ 300ms, a user usually obtains multiple groups of distance estimates each second. For a more accurate estimate, we collect 4 contiguous groups and compute 5 iterations for each group.

Looking at the standard deviation of particle distributions, and comparing to the result under 10 iterations (Fig. 3(e)), we can tell that particles converge with 5 iterations. However, when only 2 beacons are present, the result does not converge, mainly because the position cannot be uniquely determined geometrically with 2 beacons. With 3+ beacons, the result converges with an error around 0.5m. We also run tests with an increase of the average distance to beacons to $\sim$ 6m, and found the final averaged error converges to $\sim$ 2m. Essentially, with more beacons around and smaller average distances to them, the error becomes smaller.

The experiments above not only provide clues to the convergence criteria, but also provides an empirical understanding of the confidence level of user position estimates. For example, if a user is repositioned by 3+ beacons, or if its average distance to beacons is small, its position estimate is closer to the ground truth with high probability. These empirical properties serve as an important basis for our upcoming discussions.

V. A Global View: Hidden Markov Model Representation

In the previous section, we have shown the method of computing an unknown position based on the locations of nearby beacons and users, whose positions are assumed to be calculated beforehand. However, if none of the positions are known, we would encounter a cold start problem. To avoid this problem, we place beacons at locations where the users are most likely to pass by for the location information to properly propagate from known points to unknown places.

Despite these efforts, we found that the hurdle lies fundamentally in the constraints that each user can only observe nearby BLE devices, thus the impact of the observations is local. Can we expand such a local view to a more global view including all the users, in order to more accurately infer each user's position? The answer to this question is our proposed statistical crowdsourcing framework, to be presented in this section.

A. A Graphical View

We describe Tack by using a graph $\mathcal{G}$. In $\mathcal{G}$, we assume there are a total of M users or beacons scattered in the region, and each of them is represented by a node with its probability distribution. Let $\mathbf{z}_i = (x_i, y_i)$ denote the location distribution of user i or the known position of a fixed beacon. For any user i, it scans a noisy distance observation D_{ij} from its neighbor user j, or does not observe j at all. Each location variable $\mathbf{z}_i$ is associated with a prior distribution $p(\mathbf{z}_i)$, which represents the location distribution over the planar region $R = \{(x, y) \in \mathbb{R}^2 : a_1 \leq x \leq a_2, b_1 \leq y \leq b_2\}$. Node $\mathbf{z}_i$ and $\mathbf{z}_j$ are related to each other by the observed distance between them, so we should describe them in pairs.

For ease of computation, we use a particles-based sampling technique to represent continuous pairwise conditional dependencies. To avoid confusion, we use S_i to denote an arbitrary particle of user i, and S_{ir} to denote the rth particle of user i. Let the distance between particle S_i and S_j be $\bar{D}_{ij}$, i.e., $\bar{D}_{ij} = ||S_i - S_j||$. We assume the distance estimate error $D_{ij} - \bar{D}_{ij}$ follows a normal distribution with zero mean, which respects the results in Sec. III. The probability that the observed distance represents the true distance between user i and j, given the positions of two particles S_i and S_j, is:

$$p(D_{ij}|S_i, S_j) = \frac{1}{\sqrt{2\pi}\sigma} \exp(-\frac{(\bar{D}_{ij} - D_{ij})^2}{2\sigma^2}), \tag{6}$$

when D_{ij} is observed. If user i does not observe j, the probability distribution should preclude area where j is. Let $D_{\max}$ be the maximum range that BLE signal reaches, then

$$p(\neg D_{ij}|S_i, S_j) = 1 - I(\bar{D}_{ij} < D_{\max}), \tag{7}$$

when D_{ij} is not observed.

The probabilities of Eqn. (6) and (7) are the probabilities before normalization. $p(D_{ij}|S_i, S_j)$ can be seen as a discrete representation of the pairwise potential $p(D_{ij}|\mathbf{z}_i, \mathbf{z}_j)$. This pairwise potential represents the probability distribution of the observed distance given the two users' positions, and will be used in our Hidden Markov Model (HMM).

B. Hidden Markov Model

With the notations in Sec. V-A, all users' positions at time t are considered collectively state $\mathbf{Z}_t$ of the system. If we only consider the pairwise distance observations at each time t, then all users' positions at time t are independent of their respective positions from time 1 to $t-2$ given their state at time $t-1$. Thus we could describe each user's trajectory with a Markov chain. However, since each user's position is not directly visible, we introduce a *Hidden Markov Model* to describe our system.

Considering that the observation of the system often involves multiple users rather than a single user, we cannot use multiple independent Markov chains to describe Tack. Instead, we use a vectorized state $\mathbf{Z}_t = \{\mathbf{z}_{1,t}, ..., \mathbf{z}_{M,t}\}$ to describe users' states at time t in coalesce, with each variable representing each user's position. The distance observations are denoted by $\mathbf{D}_t$, which essentially consists of all pairwise distances detected between users.

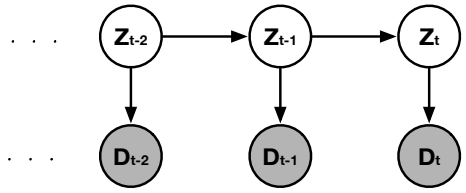


Fig. 4. Tack: A Hidden Markov Model.

Fig. 4 is an illustration of the system model, with the horizontal arrow representing the transition between states, and the vertical arrow denoting that the observation is conditioned on its state. We formulate the localization problem as maximizing the conditional probability of a state given the sequence of observations as follows:

$$\mathbf{Z}_t^* = \arg\max_{\mathbf{Z}_t} p(\mathbf{Z}_t|\mathbf{D}_1, ..., \mathbf{D}_t) = \arg\max_{\mathbf{Z}_t} p(\mathbf{Z}_t, \mathbf{D}_1, ..., \mathbf{D}_t). \tag{8}$$

Theoretically, the problem above can be solved by a variant of the Viterbi algorithm. The algorithm is a dynamic programming algorithm for searching the most likely sequence of hidden states that results in a sequence of observed events in HMMs. Let

$$\mu_t(\mathbf{Z}_t) = \max_{\mathbf{Z}_t} p(\mathbf{Z}_t, \mathbf{D}_1, ..., \mathbf{D}_t), \tag{9}$$

then the recurrence equation is

$$\mu_t(\mathbf{Z}_t) = \max_{\mathbf{Z}_t} p(\mathbf{D}_t|\mathbf{Z}_t) p(\mathbf{Z}_t|\mathbf{Z}_{t-1}) \mu_{t-1}(\mathbf{Z}_{t-1}). \tag{10}$$

In hidden markov model terms, $p(\mathbf{D}_t|\mathbf{Z}_t)$ is the *Emission Probability* representing the probability of $\mathbf{D}_t$ conditioned on $\mathbf{Z}_t$. And $p(\mathbf{Z}_t|\mathbf{Z}_{t-1})$ is the *Transition Probability*, which is the probability of the current state conditioned on its previous state.

The major challenge of this problem is to implement the dynamic programming algorithm within our framework. As we found out, the particle representation of each position, the pairwise dependencies, and the resampling procedure based on them has an innate dynamic programming structure: the *prediction* step is equivalent to computing the joint probability distribution of $p(\mathbf{Z}_t|\mathbf{Z}_{t-1})\mu_{t-1}(\mathbf{Z}_{t-1})$ by updating each particle using Eqn. (2). The *update* step is to calculate the conditional likelihood of the *Emission Probability*, which is

$$p(\mathbf{D}|\mathbf{z}_{i,t}) = \prod_{j \neq i} p(\mathbf{D}|\mathbf{z}_{i,t}, \mathbf{z}_{j,t}), \forall i, \tag{11}$$

by combining Eqn. (6) and (7). Last but not the least, *resampling* particles' weights can be considered as multiplying the likelihood to the probability distribution.

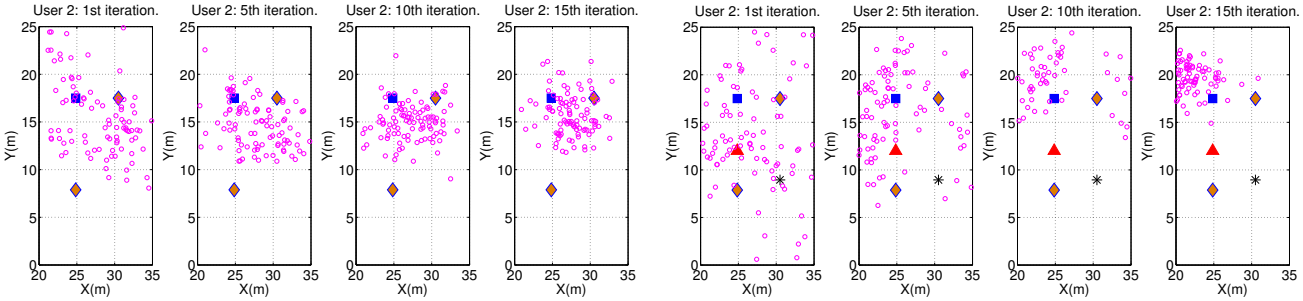
Different from the augmented particle filters in Sec. IV-A, we consider the joint probability of all user positions with respect to the observed distances in between them. Note that in the recurrence equation, the *Transition Probability* does not involve any crossing term of more than one entry, and can thus be computed locally on each user's phone. The *Emission Probability* deals with more than one entry, so for every time t, each user needs to collect all other users' data to update its position distribution using Eqn. (10).

C. Resampling with A Global View

One may naturally wonder what the advantage of resampling with a global view is, as opposed to using the local view only. We show the effect from two aspects: first, having global information improves localization accuracy; second, global information solves the "cold start" problem, *i.e.*, even if no one knows its exact position at the beginning, a global view can quickly narrow down each user's possible location range.

We run a few field tests to verify that global information helps to improve positioning accuracy. In an office of $8\text{m} \times 12\text{m}$, which is similar to a typical conference room, we place two beacons and run the positioning algorithm with each particle's behavior recorded. We then repeat the experiment by adding two more virtual beacons. Fig. 5(a) and 5(b) show the respective results with particles in each iteration represented by pink circles. Two diamonds denote two fixed beacons, while a triangle, square, and star represent the ground truth locations of User 1, 2, 3, respectively. We found that with additional two other users, the particles of User 2 converges more quickly towards its ground truth position. The result shows User 2's performance indeed improves in the presence of other users. Note that in all the tests, user positions are not known beforehand, so User 1 and 3 are not equivalent to beacons. This experiment confirmed our motivation that a global view helps propagate the repositioning effect to devices that are farther away.

Compared to our previous local approach in Sec. IV, the global view method does not assume that locations of nearby users are known. In other words, the method can still improve one's location estimate even if it observes no other users around. To verify the effect, we run a simulation on a 50×50 floor plan, with 4 fixed beacons, and 10 roaming users for 5 time slots. We compare three methods: resampling with fixed beacons, implying that each user only uses positions of beacons that it encountered to reposition itself; resampling with a local view, in that each user adopts the positions of both encountered beacons and users to reposition itself; and



(a) The way that User 2's particles converge when only 2 beacons are presented.

(b) The way that User 2's particles converge when 2 beacons and 2 other users are presented.

Fig. 5. Position estimates are improved with a global view for User 2.

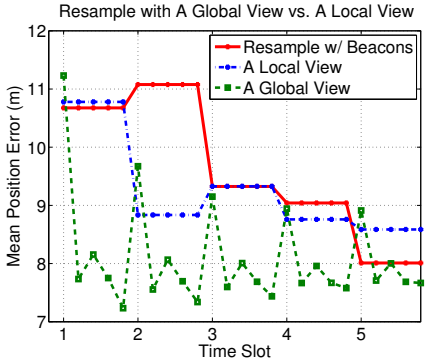


Fig. 6. A global view solves the "cold start" problem.

resampling with a global view, which collects information about all other users to assist localization.

Fig. 6 illustrates the average position error for 5 contiguous time slots using the three methods. For resampling with a global view, the figure shows the error per iteration per time slot. It is obvious that with all three methods, the error is highest in the first time slot, and gradually decreases. This is due to the fact that no prior information is obtained by any user at the outset. Within each time slot, the three methods show approximately the same position error at the beginning, except that the error of resampling with a global view decreases dramatically after a few iterations.

Essentially, at each update, resampling with a local view only provides the user whatever the current position estimate is for every encountered user – no matter the estimate is updated in the current time slot or not. This is because the user can only collect the estimates of those users that it encounters. By resampling with a global view, on the other hand, we are able to iterate through each user several times in a single update, since the location estimates of all other users are provided. Thus, as we can tell from Fig. 6, comparing to resampling with only a local view, the global view not only improves accuracy but also solves the "cold start" issue.

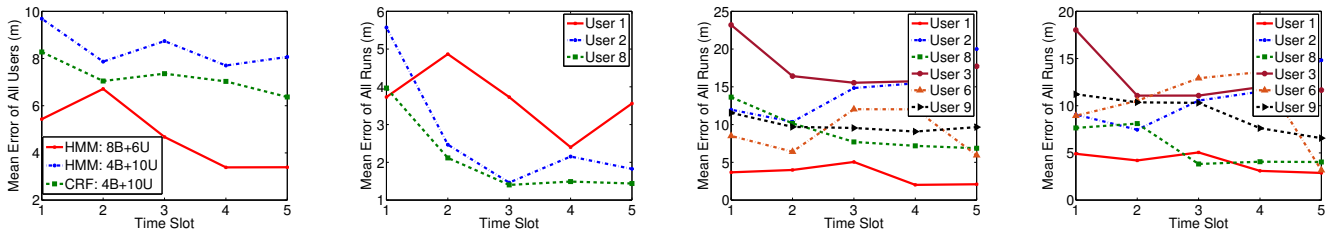
From our small-scale field tests and simulations, we found that resampling with a global view can improve localization performance from the perspective of accuracy and efficiency. However, such advantages are not obtained for free. A drawback of this approach lies in its implementation: a user will need all other users' data to compute its own position estimate, which incurs communication overhead. The scale of the particles from all the users may also introduce a significant computation overhead. We will introduce our unique implementation technique to address this challenge in Sec. VII-B.

VI. A Spatio-temporal View: Conditional Random Fields

In our experiments, we found that both dead reckoning and RSS-based observation introduced a substantial amount of noise to the localization system. Insights gained from our experiments implied that the most effective way to eliminate such noise is through immediate contacts with beacons or users with significant prior knowledge about its location. To improve the overall localization accuracy, a proper inference mechanism should be designed to maximize the influence of those positions with significant priors, and effectively propagate their information to other nodes in the system.

We run a series of simulations to verify this point. First, in the setting of 50×50 floor plan, we run the HMM algorithm as shown in Sec. V with 4 fixed beacons and 10 roaming users without any prior knowledge. Among these users, we replace User 3, 5, 6, 9 with beacons located at the same coordinates but with the positions known as prior. The resulting positioning error is shown in Fig. 7(a), 7(b) and 7(c), where 8 beacons and 6 users represent the scenario after the replacement. Obviously, when the locations of more beacons are known, the overall error is smaller. Fig. 7(b) and 7(c) show the average positioning error for each user throughout time slots. User 1, 2, 8 remain after replacement while User 3, 6, 9 are replaced in Fig. 7(c). Examined more closely, the positioning results of User 2 and User 8 deteriorate with crowdsourcing if no prior knowledge of users' positions is learned beforehand. Other users' positioning results are not shown due to space constraints.

Knowing that the significant prior can greatly improve localization accuracy, how should we utilize it in Tack? The problem lies in three aspects: first, how to define nodes with significant prior information; second, how to increase the number of nodes with such auxiliary information in the system; and finally, how to utilize that extra bit of information to improve positioning accuracy. We will answer each question respectively.



(a) The Overall Effect (b) HMM for 8 Beacons and 6 Users (c) HMM for 4 Beacons and 10 Users (d) CRF for 4 Beacons and 10 Users

Fig. 7. Beacons introduce significant prior information; how auxiliary information improves accuracy ("B" and "U" represent beacons and users respectively; partial results are shown).

Since the nodes with prior known are sparsely scattered in the venue, or along a user's trajectory, it would be beneficial to jointly consider time and space for increasing the number of such nodes in the system. Thus, we introduce a more powerful and general graphical tool called *Conditional Random Field* (CRF) to describe our localization system from a theoretical perspective. With a spatio-temporal view, we would be able to incorporate more features that help localization.

But the advantage is gained without its unique challenge. In the previous HMM-based approach, due to the "memoryless" property of Markov chains, one computes its position only based on the current state and one state before. With CRF, we essentially incorporate more time-related features when inferring user positions. To achieve that, we need to file the history data collected from all users within a certain window. We would elaborate on the features and the implementation detail in this section.

A. Conditional Random Fields

Conditional random fields (CRFs) can be considered generalization of HMMs. They are undirected graphical models that encode a conditional probability distribution $p(\mathbf{Z}|\mathbf{D})$ using a given set of features. While HMM treats the observations across different times as conditionally independent given the position states, CRF needs no independence assumption between the observation variables. CRFs have been widely used in applications such as robot navigation and speech recognition. By eliminating the independence assumptions between observations, the model is more flexible and capable of describing many more features.

In this section, we first transform our system HMM model to a CRF model and show how these features are implemented in Tack.

From HMM to CRF. Different from HMMs, CRFs are more flexible in describing systems with arbitrary features. Each feature takes in as input: the observations across all times $\mathbf{D}$, the current time t, the current state $\mathbf{Z}_t$, and the previous state $\mathbf{Z}_{t-1}$. To compute the conditional probability $p(\mathbf{Z}|\mathbf{D})$ on an undirected graph, we use cliques potentials rather than the product of conditional distributions as in a directed model. A clique is a subgraph of which every pair of nodes is connected by an edge. In CRF presentation, we consider that all observations are represented as an entirety $\mathbf{D}$, which is linked with any state node like in Fig. 8. In that figure, node $\mathbf{Z}_{t-1}$, $\mathbf{Z}_t$, and $\mathbf{D}$ form a clique.

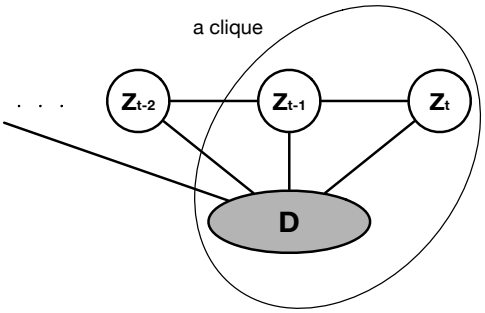


Fig. 8. Tack: A Conditional Random Field Model.

Let $C = \{\mathbf{z}_c\}$ be the set of cliques in $\mathcal{G}$. Then CRFs define the conditional probability of state variables as follows:

$$p(\mathbf{Z}|\mathbf{D}) = \frac{1}{P(\mathbf{D})} \prod_{c \in C} \Phi(\mathbf{D}, \mathbf{z}_c). \tag{12}$$

where Φ is the potential function defined over a clique, and $P(\mathbf{D})$ is the partition function that is a normalization factor over all variables states. We assume the potentials factorize according to a set of feature functions $\{f_k\}$, to each of which a real-valued weight λ_k is attached. Thus,

$$\Phi(\mathbf{D}, \mathbf{z}_c) = \prod_{k=1}^{K} \exp(\lambda_k f_k(\mathbf{D}, \mathbf{z}_c)). \tag{13}$$

To extend HMM with the time dimension, we "unroll" the model in time and present Tack using an undirected graph $\mathcal{G}$ as in Fig. 9. All clear circles represent state nodes which describe position distribution, and the grey circles are observations nodes. HMM requires observations are independent of each other given the state, thus it is unable to describe feature like user displacement $\mathbf{D}_{i,t-1}$ across different times.

Since all states are connected to a common observation, a typical clique contains $\mathbf{D}$, $\mathbf{Z}_{t-1}$ and $\mathbf{Z}_t$. The conditional probability can be further derived as:

$$p(\mathbf{Z}|\mathbf{D}) \propto \prod_{t=1}^{T} \prod_{k=1}^{K} \exp(\lambda_k f_k(t, \mathbf{D}, \mathbf{Z}_{t-1}, \mathbf{Z}_t)). \tag{14}$$

Our objective is to find the set of states $\mathbf{Z}$ that maximizes the conditional probability in Eqn. (14) by converting HMM into CRF without losing the advantage of HMM. To incorpo-

rate both *Transition* and *Emission* probabilities into the CRF model, we introduce two corresponding features:

$$f_1(t, \mathbf{D}, \mathbf{Z}_{t-1}, \mathbf{Z}_t) = I(\mathbf{D}_{t-1})I(\mathbf{Z}_{t-1})I(\mathbf{Z}_t), \\ f_2(t, \mathbf{D}, \mathbf{Z}_{t-1}, \mathbf{Z}_t) = I(\mathbf{D}_t)I(\mathbf{Z}_t), \tag{15}$$

and their respective weights are:

$$\lambda_1 = \log p(\mathbf{D}_{t-1}|\mathbf{Z}_{t-1}, \mathbf{Z}_t), \\ \lambda_2 = \log p(\mathbf{D}_t|\mathbf{Z}_t). \tag{16}$$

The weight of the first feature is the conditional potential of the displacement observation given positions $\mathbf{Z}_{t-1}$ and $\mathbf{Z}_t$. Essentially, the higher the likelihood, the more weight will be assigned to the position pairs. And the second feature corresponds to the likelihood of observations given certain position probability distribution.

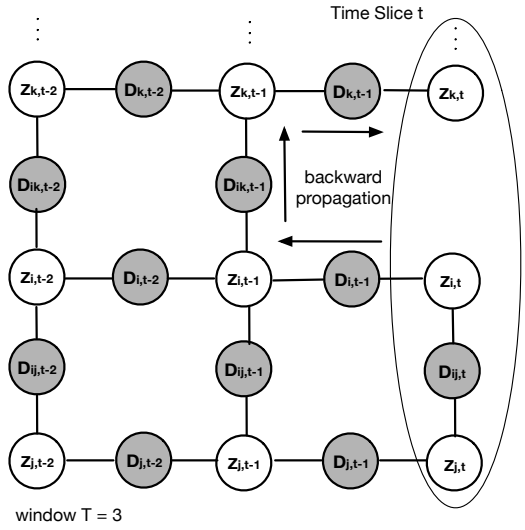


Fig. 9. Tack: A Spatio-temporal View.

Revisiting Transition Probability. The transform above from HMM to CRF is not sufficient. Note that the transition probability in HMM is the probability of the state conditioned on its previous state, which is not only invalid on an undirected graph but also precludes the possibility of backward propagation, *i.e.*, to let the current state influence the past state.

To make the feature more expressive, we adopt pairwise potential functions to describe the bidirectional transition probability between node $\mathbf{z}_{i,t-1}$ and $\mathbf{z}_{i,t}$. We build the pairwise potential function based on the *kinematic model* using the DR results. In the kinematic model, we calculate user displacement depending on the combined results of the step counter and magnetometer. To calculate the displacement from time $t-1$ to t, we accumulate the displacement of each step in between:

$$\begin{pmatrix} \Delta x_i \\ \Delta y_i \end{pmatrix} = \begin{pmatrix} \sum_k \Delta x_i^k \\ \sum_k \Delta y_i^k \end{pmatrix} = \begin{pmatrix} \sum_k \beta_i \cos(\theta + \alpha_i^k) \\ \sum_k \beta_i \sin(\theta + \alpha_i^k). \end{pmatrix} \tag{17}$$

β_i is the average stride length, and α_i^k is the heading orientation per step of user i.

We assume the user displacement in Eqn. (17) follows the Gaussian distribution around the true value according to the results in Sec. III. Based on that, we compute the pairwise potential between node $\mathbf{z}_{i,t-1}$ and $\mathbf{z}_{i,t}$. As usual, a discrete representation of the distribution is adopted in the form of pairwise potential between two particles $S_{i,t-1}$ and $S_{i,t}$ as follows. Again, $S_{i,t}$ represents an arbitrary particle of user i at time t.

$$p(D_{i,t-1}|S_{i,t-1}, S_{i,t}) = \frac{1}{2\pi\sigma^2} \exp(-\frac{(x_{i,t} - x_{i,t-1} - \Delta x_i)^2}{2\sigma^2}) \\ \times \exp(-\frac{(y_{i,t} - y_{i,t-1} - \Delta y_i)^2}{2\sigma^2}) \tag{18}$$

By using Eqn. (18), we are allowed to express the transition probability on the undirected graph. We will show the power of this expression in the following section.

Reviewing Fig. 9, Conditional dependencies between variables representing different users at the same time slice can be expressed in the form of pairwise relationship as depicted in Eqn. (6). The edge across different time slices is formulated as the pairwise potential function Eqn. (18). The figure shows an example where the time window is 3.

Equipped with CRF, we not only preserve the advantage of HMM by directly importing the transition and emission probabilities, but also are able to describe more spatial or temporal features.

B. Confidence and Trend

We now try to answer the question of how to determine nodes with more significant prior than others, and how to integrate the feature within the CRF framework.

By significant prior, we essentially wish that a node with a higher confidence about its position will influence a node with a lower confidence, but not the other way around to prevent the estimation error from propagating. For example, a node that is close to a fixed beacon has a higher confidence about its position estimate than a node not observing anything. More specifically, having examined the property in Sec. IV-C, we can distinguish the nodes with a higher confidence by using criteria such as the number of nearby beacons and the distance to the beacons. In Tack, we implement more than such an intuition. Each user maintains a weight account to estimate its confidence level. The weight account is assigned a high value whenever the user gets repositioned with nearby fixed beacons, representing high confidence about its estimated position; one point is deducted from the account when the user takes a step implying the confidence level reduces as the user position drifts.

The "confidence" feature is special in CRF in that we do not usually specify the inference direction over an undirected graph. However, in the inference algorithm to be illustrated later, we will show that the feature can be implemented without violating the problem structure. We define the feature as follows. Letting $W_{i,t}$ be the weight left in the account of user i at time t,

$$\exp(f_{k \to i}(t, \mathbf{D}, \mathrm{z}_{i,t}, \mathrm{z}_{k,t})) = I(W_{k,t} > W_{i,t}), \\ \exp(f_{t-1 \to t}(t, \mathbf{D}, \mathrm{z}_{i,t}, \mathrm{z}_{i,t-1})) = I(W_{i,t-1} > W_{i,t}).$$

Our simulations (with the same settings as our previous ones) show the effect before, indicated by HMM, and after, represented by CRF, the "confidence" feature is implemented. With some prior knowledge about each user's estimates, the overall error is reduced by almost 2m, as shown in Fig. 7(a). Examining more closely, User 2, 3 and 8 respectively improves accuracy without deteriorating other users' accuracy, if we compare Fig. 7(c) and 7(d).

Apart from the "confidence" feature, we can do far more with CRF. For example, BLE trends measured from continuous BLE signals across time slots can be considered as a feature. A user's position change should be consistent with the wireless signal trend: when the signal trend shows that a user is approaching a beacon, its trajectory cannot go the opposite way. This feature is derived from our empirical observation that the trend of the wireless signal strength is far more reliable than the strength itself, and existing work [11] echos our approach. For user i and fixed beacon j, the feature function is defined as follows:

$$\exp(f_3(t, D_{ij,t}, D_{ij,t-1}, \mathbf{z}_{i,t}, \mathbf{z}_{i,t-1})) = I(\frac{sign(D_{ij,t} - D_{ij,t-1})}{sign(||\mathbf{z}_{i,t} - \mathbf{z}_j|| - ||\mathbf{z}_{i,t-1} - \mathbf{z}_j||)} > 0).$$

where $\mathbf{z}_{i,t}$ represents the position of user i at time t and $\mathbf{z}_j$ is the known position of the fixed beacon j. When user i is observed to approach a fixed beacon, the feature assigns a value of 1 if its position estimate reflects the approaching trend, and 0 otherwise. The same feature applies when user i goes the other way around.

Such a trend feature takes advantage of the powerful presentation of CRF over HMM: it is able to describe overlapping portions of the observation sequence. An HMM with such overlapping feature is no longer a proper generative model, nor is the likelihood function correct.

Backward Propagation. One may naturally wonder what the benefit is to take the historical trajectories and observations into account in CRF while our system possesses the Markov property and HMM seems to be a fit. In fact, historical data helps to improve the overall positioning accuracy by increasing the number of nodes on the undirected graph that have significant prior distributions. We achieve this through forward and backward propagation between nodes at different time instances.

Fig. 9 gives such an example: when user i encounters a fixed beacon at time t, it successfully corrects its position and introduces significant prior knowledge into the undirected graph. If we only consider adopting the approach in Sec. V, the corrected position $\mathbf{z}_{i,t}$ may never affect $\mathbf{z}_{k,t}$ for the broken link in between. Even if user k is somehow connected with user i through other users, $\mathbf{z}_{k,t}$ can only be influenced indirectly by $\mathbf{z}_{i,t}$ via other nodes. By incorporating historical data, nodes can affect each other through different paths. For example, the message from $\mathbf{z}_{i,t}$ can propagate to $\mathbf{z}_{i,t-1}$ through backward induction, then $\mathbf{z}_{i,t-1}$ affects $\mathbf{z}_{k,t-1}$ through pairwise potential, and finally, $\mathbf{z}_{k,t}$ receives the message from $\mathbf{z}_{k,t-1}$ to adjust its distribution through forward induction.

The CRF model is able to correct location estimates in the past, and that correction helps to deduct a more accurate current estimates overall, even in the case that the users are not currently in contact with each other. By incorporating the history information, the inference algorithm over the graphical model is able to yield a more accurate result for each user.

C. Inference on CRF

In our problem, since we aim at estimating the position variables for all users, which are represented by particles, with all the features, the combinations over such a large domain is exponential. This makes exact inference computationally intractable for Tack, especially in the case that location tracking is needed. Therefore, we choose to apply an approximate inference algorithm to solve it.

We wish to solve the problem of computing the unobserved variables $\mathbf{z}_{i,t}$ for all users at different time points within the window to maximize the joint conditional probability of Eqn. (12). The problem can be solved by inference, which is a particular type of learning, aiming at searching the instances of the hidden variables that maximizes their probability conditioned on all observed data points. Learning a CRF can be done by any inference algorithm for undirected models, such as iterated conditional modes, Gibbs sampling, loopy belief propagation, etc.

We choose to use *Gibbs sampling* as our approximate inference algorithm due to the Markov property of the system. The method is a Markov chain Monte Carlo (MCMC) algorithm for obtaining a sequence of samples which are approximated from a specified multivariate probability distribution. It generates a Markov chain of samples, each of which is correlated with nearby samples.

Initially, we randomize all users' positions over a planar region, and then repeat the following procedures for multiple iterations until convergence. In each iteration, we compute the probability distribution of each node conditioned on all other nodes in the graph, and then sample from the distribution. As defined by the local Markov property, the conditional probability of $p(\mathbf{z}_{i,t}|\mathbf{D}, \mathbf{Z}\backslash\mathbf{z}_{i,t})$ can be expressed as the probability conditioned on its neighbors $\mathcal{N}_{\mathbf{z}_{i,t}}$. For example, in Fig. 9, the neighbors of node $\mathbf{z}_{i,t-1}$ are $\mathbf{z}_{i,t-2}$, $\mathbf{z}_{i,t}$, $\mathbf{z}_{j,t-1}$, and $\mathbf{z}_{k,t}$. The conditional probability is computed based on the feature functions, and we wish to find the most likely joint states probabilities. For each node $z_{i,t}$, we compute:

$$\begin{aligned} p(\mathbf{z}_{i,t}|\mathbf{D}, \mathbf{Z}\backslash\mathbf{z}_{i,t}) &= p(\mathbf{z}_{i,t}|\mathbf{D}, \mathcal{N}_{\mathbf{z}_{i,t}}) \\ \propto \prod_{\mathbf{z}_{j,s}\in\mathcal{N}_{\mathbf{z}_{i,t}}} \prod_k \exp(\lambda_k f_k(\mathbf{D}, \mathbf{z}_{i,t}, \mathbf{z}_{j,s})), \\ \forall i, j \in [1, M], \forall t, s \in [1, T]. \end{aligned} \tag{19}$$

Then we sample $\mathbf{z}_{i,t}$ from $p(\mathbf{z}_{i,t}|\mathbf{D}, \mathbf{Z}\backslash\mathbf{z}_{i,t})$. We repeat the probability computation and sampling procedure for each hidden state node on $\mathcal{G}$ until convergence. Upon convergence, the joint states probability of all nodes is maximized, and the probability distribution for each node represents the most likely distribution of the positions.

VII. Implementation and Evaluations

We have conducted both large-scale simulations and real-world experiments using iOS devices, based on our real-world implementation of Tack.

A. Simulations

In simulations, we attempt to emulate the real-world crowds as much as possible to reveal how Tack works against large crowds with different approaches. We use a random waypoint model to generate user traces on a 50×50 floor plan for 20 contiguous time slots, with a minimum velocity of 1.0, a maximum velocity of 4.0, and a maximum waiting time of 2.0. Fig. 10(a) illustrates an example of visualized user trajectories throughout 20 time slots for 15 users. Based on those traces, we further generate the displacement, user-to-user, and user-to-beacon distance observations, according to each respective model in Sec. III. For instance, with respect to the user-to-beacon distance observation, we draw the observed value from a Gaussian distribution centered around the true value, with a standard deviation linearly increasing with the true distance, and a maximum observable range 13.0. Other observations are generated accordingly as well.

As we try to understand how the crowd sizes and different methods affect the position error statistically, we test 4 crowd sizes — 7, 15, 30 and 45 users, using three methods — the HMM algorithm, and the CRF algorithm with window size being 3 and 5, respectively. In all the settings above, we place 6 beacons to each, and each setting is tested for $30+$ runs.

TABLE I
Mean / Standard Deviation of Errors (meters)

	7 Users	15 Users	30 Users	45 Users
HMM	3.63/2.14	4.02/1.97	4.45/1.78	7.16/3.13
CRF (window = 3)	3.55/1.76	3.78/1.90	3.63/1.54	4.77/1.95
CRF (window = 5)	3.54/1.85	3.69/1.50	3.55/1.65	4.73/1.93

Results. Due to space constraints, not all our results can be included in the figures. Readers may refer to Table I for complete results. Fig. 10(b) and 10(c) show the position error at each time slot for different crowd sizes when applying different methods.

The general trend is that the CRF algorithm has better accuracy than HMM, and the accuracy improves with larger window sizes. Comparing the errors across different crowd sizes, we surprisingly find that the error does not necessarily decrease as the crowd size gets larger where more data fed into the system. For HMM, the error increases as the crowd size gets larger. For CRF, the error almost remains the same when the user number varies from 7 to 30, but increases by almost 1 meter when the crowd size is 45. The reason is that with more data input, more errors are introduced into the system. The pairwise dependencies between users propagate not only the genuine location information but also the errors in the system. Fortunately, CRF can prevent such error propagation to some extent by incorporating features and side observations. Thus, we observe a relatively mild increase in errors with CRF as the crowd size increases.

Fig. 10(d) and 10(e) illustrates the error distributions for varying crowd sizes with different methods. For HMM, 70% of errors are within 4.0m, 4.1m, 5.1m and 9.8m for user number is 7, 15, 30 and 45. For CRF with a window size of 3, the corresponding results are 4.15m, 4.15m, 4.0m and 5.45m respectively. Finally, we vary the number of beacons to see how it affects the resulting accuracy. Fig. 10(f) shows the error distribution when the number of users is 30, and the running algorithm is CRF with a window size of 3. Overall, given the same setting, the more beacons, the higher the resulting accuracy. For 6, 8, and 9 beacons, 70% of errors are within 4.0m, 3.27m and 2.84m respectively. The costs of deployment should be considered when pursuing a higher localization accuracy in practice.

B. Implementation

1) A Software Framework: As a prototype system, we adopt iOS as the mobile development platform, and `Parse` server as the mobile backend for collecting and broadcasting user data. Considering the scale of our experiment is not very large (~ 10 users), we implement all computing functions on mobile devices for fast prototyping. The performance cost will be discussed in later sections.

The interface of Tack is shown in Fig. 11.The left part of the figure is our main interface. In the upper middle of the screen, the real-time location is displayed in the form of (x, y) coordinates. The steps taken are shown at the bottom left corner while the confident level sits at the bottom right corner. When the user presses the "Report" button, its current position is recorded. A user can also intuitively get its position by the moving yellow dot on the map, as shown in the right part of the figure.

We have implemented the main body of our crowdsourcing algorithm as a framework in the Swift programming language on the iOS platform, called `LocationKit`. An application can easily import `LocationKit` as a library in order to provide its indoor localization service. `LocationKit` incorporates several components: particle filter, dead reckoning, communication, and the inference algorithm. The inference algorithm is the core component built on particle filter, while particle filter relies on dead reckoning. The communication module is mainly used for communicating with the server. If the connection to the server breaks down, Tack will fall back to a local operation mode, and estimates the position based on dead reckoning and resampling according to a local view.

Fig. 12 gives an overview of the workflow on the implemented prototype system. The left part runs locally on each mobile device while the right part describes the communication with the server. At the beginning, the device registers with the server and sends its unique "name" to associate with its object ID once and for all. The device's "name" is used to distinguish from other mobile devices in crowdsourcing. With its name, a device can broadcast to others via BLE *advertiser* mode. At the same time, the device creates a `TLocation` object to inspect local location updates. To do that, `TLocation` initiates dead reckoning, *i.e.,* the `TStepCounter` class, and turn on BLE *scanner* mode to scan other BLE devices. The

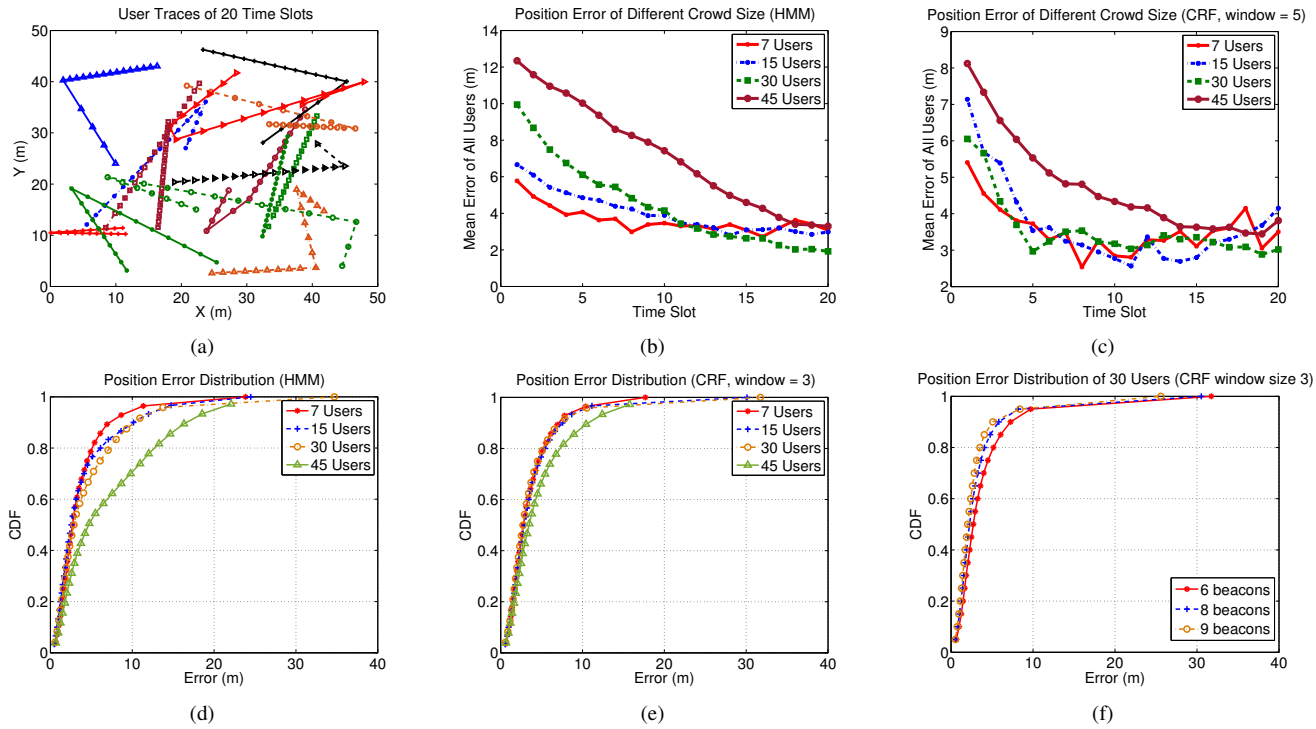


Fig. 10. Simulation Setting and Results.

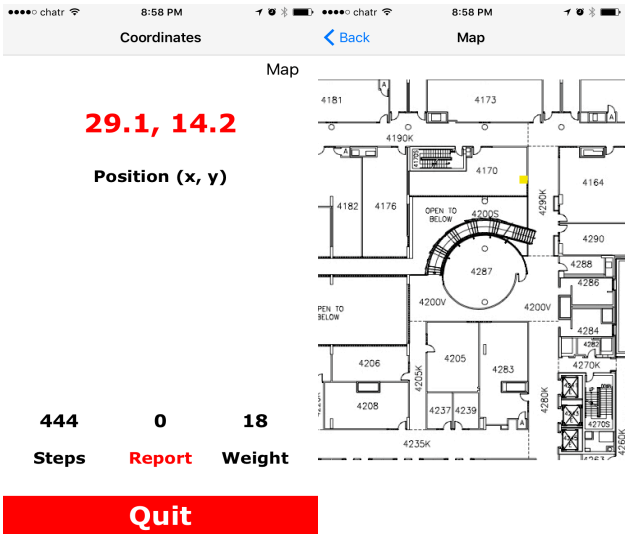


Fig. 11. Tack: interfaces.

particles of the device are supervised by the `TParticleFilter` class and the class reacts accordingly to the feedback of dead reckoning, scanning results of nearby beacons and the data pulled from the server.

Taking inputs from the dead reckoning output, observations of nearby beacons and other devices' information from the server, the MAP inference algorithm calculates the most likely position of the current user. As the inference algorithm is the core part of Tack, we will discuss the issues encountered as below.

2) Code-level Optimizations: Implementing the inference algorithm as described in Sec. VI-C seems to be easy, but one should keep in mind it's a machine learning algorithm which can be very inefficient when running on a mobile device. Thus we adopt all measures below to ensure efficiency.

Vectorization. The first issue we face is the huge sample space of particles of all users. For instance, if we assign 100 particles to each user, with 10 users the total number of particles reaches 10^3. For any two users, the number of particle pairs will be 10^4. If we execute all particle-wise operations in loops, each iteration will take more than 1 second to finish. This is far too slow for the localization service, not to mention the intensive computation drains the battery power quickly.

Hence, we utilize the `Accelerate` framework on iOS to reduce the computation cycles. The `Accelerate` framework exposes SIMD instructions available in modern CPUs to improve the performance when working with arrays or matrices. We novelly express particles of each user in a matrix and convert all the particle-wise operations to matrices operations, so that all computation of the conditional likelihood are performed in batches. This greatly improves the computation efficiency and eventually gains at least a $10\times$ speedup.

Early termination. While processing data in batches improves the computation speed, we found the speed can be improved further. Compared to running the inference algorithm for a fixed number of iterations in each update, we terminate the computation earlier if the difference between the results in the last two iterations is smaller than a threshold.

Scaling up very small numbers. Another problem we encountered is that in calculating the conditional probability of each particle, the probability sometimes can be too small to be represented by fixed point numbers. As we compute

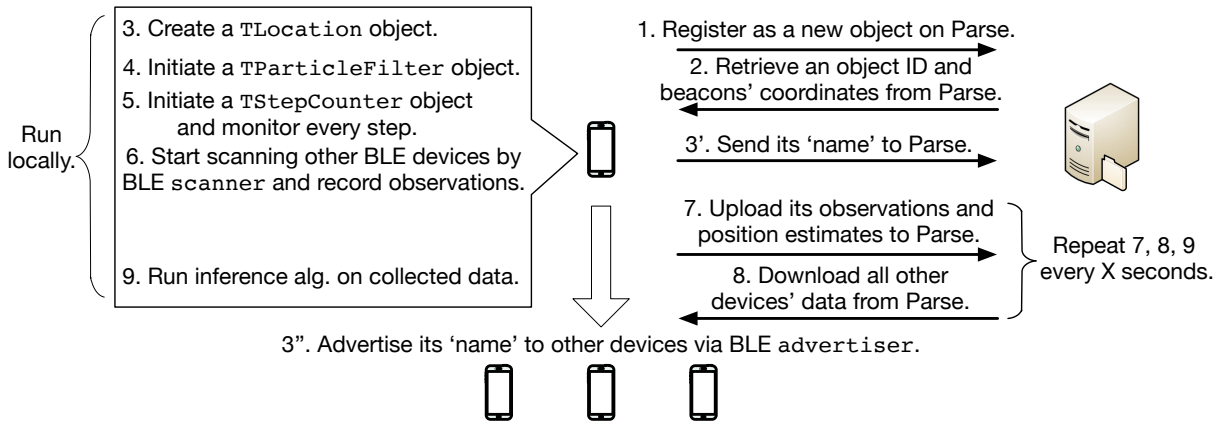


Fig. 12. The workflow of Tack.

the conditional probability in Eqn. (19), a large number of neighbors and high weights λ_k in the exponent can easily yield very small probabilities. To avoid the problem, we first compute all exponential coefficients, cap them by a constant and use the results as the new exponential coefficients in Eqn. (19). With the conditional probabilities of all particles normalized per node, this approach produces correct results without generating very small numbers.

C. Performance Evaluation

In this section, we will first show the results of our real-world experiments, and then discuss the performance overhead and other costs.

We have evaluated Tack in a hallway of $40\text{m}\times 50\text{m}$ deployed with 7 beacons, which mimics the hallway between conference rooms at a conference venue. To acquire the ground truth, we record several random user traces, and randomly place markers with an average of 1.5m between adjacent ones. The user trajectories and placement of beacons are shown in Fig. 13(a). Red dots represent beacons, and all 7 trajectories are labeled using the same colors of the users.

To verify the effectiveness of our crowdsourcing algorithm, we ask users to follow the marked trajectories and Tack will log the position estimates while the users move along these trajectories. Throughout the entire trajectory, each user would meet 2.8 other users on average. We test the system with different settings, and repeat our experiments for 5-10 runs for each setting. In our evaluation, we wish to inspect both instantaneous errors over time and average errors.

Results. By varying the number of participating users and the algorithm parameters, we have 6 different configurations of setting: the total number of participants are 5 and 7 respectively, HMM algorithm, CRF algorithm with window size being 3 and 5. All instantaneous errors are recorded and parts of the results are shown in Fig. 13(b)-13(f).

In Fig. 13(b), we compare the performance of different algorithms with the same number of participants. When 5 users roaming at the same time, the percentage of errors that are within 5m for HMM, CRF with window size 3 and CRF with window size 5 are 84%, 76%, 84% respectively. For HMM, in more than 5% of the cases, the error is as large as 10m, while for CRF with window size 5, there are almost no cases where the error is larger than 8 meters. In Fig. 13(c), we compare the impact of a different number of participants on the same algorithm, in this case, CRF with window size 5. We found the error is less than 5m in 85% of the cases with 7 users and in over 90% of the cases with 5 users.

We calculate the average errors for all different settings and summarize the results in Fig. 13(d). All average results are between 2-4m. For all three algorithms, 7-user outperforms 5-user, with an improvement of 22.5% on average. In general, CRF with larger window size has better performance than those with smaller window size. HMM has good average accuracy but is prone to larger errors in some corner cases.

For further examination, we study User 3's instantaneous error under various settings. The result is averaged over 10 runs. We found that in all cases, its error is smallest both at the start and end, where the user is the nearest to the position of a fixed beacon. The error increases as the user walks away from a beacon, and reduces in the middle where the user encounters other virtual beacons. To verify the error reduction is due to the encounter with other users, we show the instantaneous error of all users in one run in Fig. 13(f). From this figure, we found that as User 3 encounters User 4 and User 5 halfway through its trajectory, its error drops as User 4 and User 5 have recently been repositioned by fixed beacons.

Overall, our real-world experimental results are in accord with the simulations results, even with better accuracy in some cases. This is because due to the building structure and physical confinement, the actual area that the users wander about is more constrained than in simulations, so that the user trajectories tend to overlap more.

Performance costs. The localization accuracy is not achieved without any sacrifice. While more participating users can improve the accuracy, it naturally increases the computation overhead, especially when the inference algorithm runs locally. By experiments, we show how the computation overhead vary with the crowd sizes, and prove that Tack is a relatively light-weight system that would not burden mobile devices.

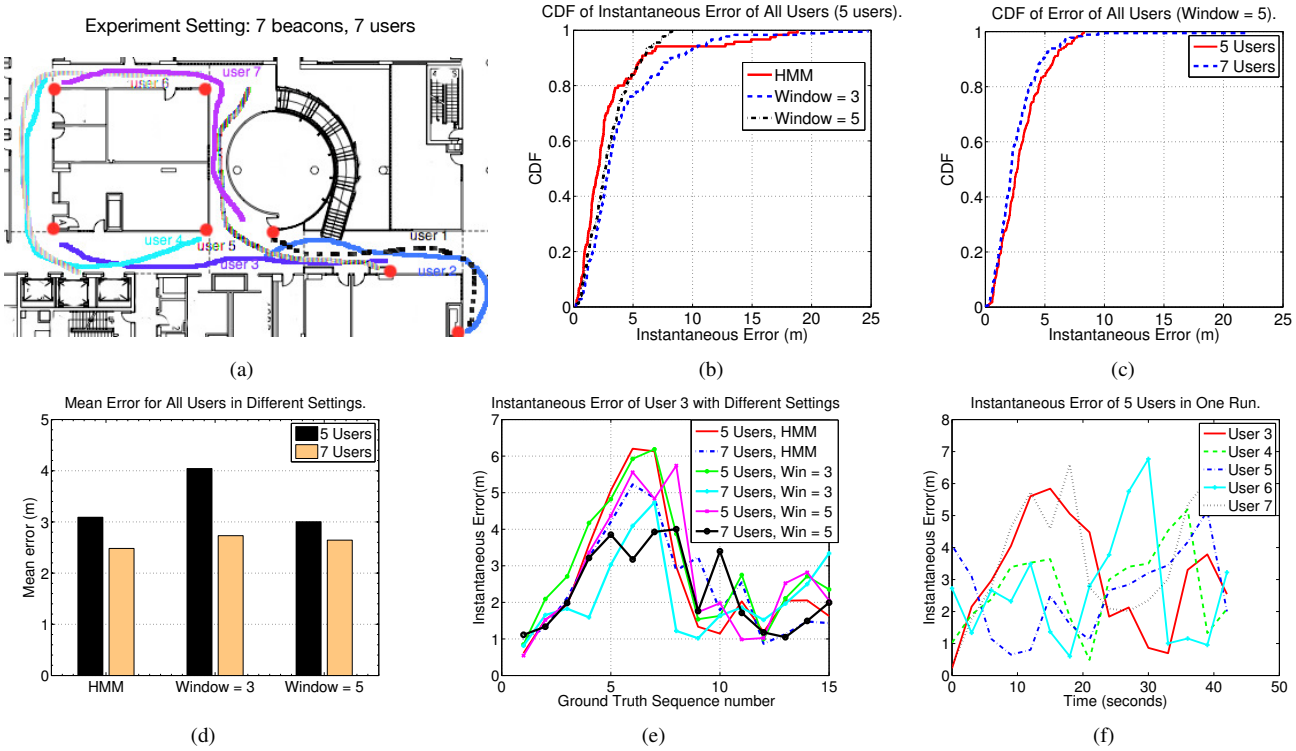


Fig. 13. User Traces and Experimental Results.

All following experiments are repeated for sufficient rounds to obtain the average value. Since the results vary according to different models and types of devices, we report the results on iPhone6S (iOS 10.2) as an example in Table II and Fig. 14. For all experiments, we reused the same setup as in the last section and choose CRF algorithm with window size 3 as the crowdsourcing algorithm.

To properly measure the energy consumption on iOS devices, we use the time period that the battery power drops by 1% as the metric. We believe this is the most accurate measurement we can obtain since Apple does not provide any interface for battery capacity, nor can we tear the device down. As the display and other factors also consume the battery power, we set a control group by running the same interface without the localization functions turned on. As a result, the control group runs 203.75s on average before the battery power drops by 1%. To compare with other localization systems, we also run a group of experiments where the devices continuously scanning for WiFi, which is the basis for WiFi fingerprinting.

To gauge the localization latency, we use as a metric the duration between the time that a user uploads its prior estimate and observations to the server and the time that the user finishes running the inference algorithm and update its location.

As Table II shows, the energy consumption is reasonable in Tack. Compared to the idle states, the time period that the battery power dropping 1% reduces by 2.7%, 9.2%, 15.5% and 24.5% respectively for 1, 4, 7, 10 users. In the case of WiFi scanning, the corresponding reduction is 60.43%. On one hand, Tack is obviously more energy-efficient than WiFi-based localization systems. On the other hand, the results show

TABLE II
ENERGY CONSUMPTION AND LATENCY

Mean/(STD)	WiFi	1 User	4 Users	7 Users	10 Users
1%-dropping time (s)	80.62	198.33	185.0	172.25	153.75
Localization latency (s)	–	0.34/0.14	0.44/0.11	0.58/0.13	0.73/0.18

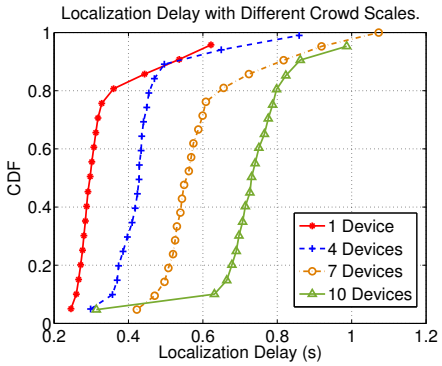


Fig. 14. Localization latency with different crowd sizes.

that the battery power consumption is mostly introduced by the increased computation complexity, which can be easily migrated if we deploy the inference algorithm on the server rather than the mobile device.

Table II also shows that the mean of localization latency grows almost linearly with the number of participants, which corresponds to the power consumption performance. In 80% of all cases, the localization delay is within 0.36s, 0.46s, 0.65s, and 0.80s respectively for 1, 4, 7, 10 users, and all

cases are within 1 second, as shown in Fig. 14. We think the energy cost and latency cost are acceptable for an indoor localization service. But for a very large crowd (30+ users), it is recommended to deploy the inference algorithm to the server to alleviate the burden from mobile devices.

Infrastructure costs. We would like to compare the infrastructure costs with a typical WiFi RSS-based indoor localization scheme. Sorour *et al.* [12] builds a simultaneous localization and radio mapping system and experiment it at the same location as we are. They achieved 2 to 4 meters of accuracy by using five Linksys access points with each costing over 80$. Tack, deployed in the same area, achieved the same accuracy level with 7 Estimote beacons with each only 10-20$. Besides, the installation of beacons is far easier than access points. They are portable and lasts for two years without external power sources.

To sum up, our experimental results have shown that Tack can effectively provide accurate indoor localization for different scales of crowds. It is light-weight to run on a mobile device. For larger crowds, one can easily migrate a part of the computation to the server. More importantly, it is very cheap and convenient to deploy, thus is suitable for ephemeral event locations, such as a conference.

VIII. Conclusion

Tack is one of the few practical, easy-to-deploy indoor localization systems building around the new wireless technology BLE. We exploit several techniques such as dead reckoning, virtual beacons, and contacts over BLE, together with the crowdsourcing algorithms to create a software framework. We improve the localization performance progressively through both theoretical analysis and field tests. Implemented as a framework in the Swift programming language, Tack's performance is extensively evaluated on the iOS platform in real-world experiments. Overall, Tack achieves an accuracy of 2-4 meters indoors when 7 beacons are deployed in an area of $40\text{m} \times 50\text{m}$. Compared to other indoor localization systems on smartphones, Tack is accurate, energy-efficient and less costly.

References

[1] iBeacon for Developers. https://developer.apple.com/ibeacon/. Accessed: 2015-07-21.

[2] H. Abdelnasser, R. Mohamed, A. Elgohary, M. F. Alzantot, H. Wang, S. Sen, R. R. Choudhury, and M. Youssef. SemanticSLAM: Using Environment Landmarks for Unsupervised Indoor Localization. *IEEE Transactions on Mobile Computing (TMC)*, 15(7):1770–1782, 2016.

[3] P. Bahl and V. N. Padmanabhan. RADAR: An In-building RF-based User Location and Tracking System. In *Proc. IEEE INFOCOM*, volume 2, pages 775–784, 2000.

[4] K. Chintalapudi, A. Padmanabha Iyer, and V. N. Padmanabhan. Indoor Localization without the Pain. In *Proc. of 16th ACM MobiCom*, pages 173–184, 2010.

[5] I. Constandache, X. Bao, M. Azizyan, and R. R. Choudhury. Did You See Bob? Human Localization Using Mobile Phones. In *Proc. 16th ACM MobiCom*, pages 149–160, 2010.

[6] I. Constandache, R. R. Choudhury, and I. Rhee. Towards Mobile Phone Localization without War-driving. In *Proc. IEEE INFOCOM*, pages 1–9, 2010.

[7] S. He, W. Lin, and S.-H. Chan. Indoor Localization and Automatic Fingerprint Update with Altered AP Signals. *IEEE Transactions on Mobile Computing (TMC)*, 2016.

[8] J. Jun, Y. Gu, L. Cheng, B. Lu, J. Sun, T. Zhu, and J. Niu. Social-Loc: Improving Indoor Localization with Social Sensing. In *Proc. 11th ACM Conference on Embedded Networked Sensor Systems (SenSys)*, page 14, 2013.

[9] A. Rai, K. K. Chintalapudi, V. N. Padmanabhan, and R. Sen. Zee: Zero-effort Crowdsourcing for Indoor Localization. In *Proc. 18th ACM MobiCom*, pages 293–304, 2012.

[10] I. M. Rekleitis. A Particle Filter Tutorial for Mobile Robot Localization. *Centre for Intelligent Machines, McGill University, Tech. Rep. TR-CIM-04-02*, 2004.

[11] G. Shen, Z. Chen, P. Zhang, T. Moscibroda, and Y. Zhang. Walkie-Markie: Indoor Pathway Mapping Made Easy. In *Proc. 10th USENIX Conference on Networked Systems Design and Implementation (NSDI)*, pages 85–98, 2013.

[12] S. Sorour, Y. Lostanlen, S. Valaee, and K. Majeed. Joint Indoor Localization and Radio Map Construction with Limited Deployment Load. *IEEE Transactions on Mobile Computing (TMC)*, 14(5):1031–1043, 2015.

[13] H. Wang, S. Sen, A. Elgohary, M. Farid, M. Youssef, and R. R. Choudhury. No Need to War-drive: Unsupervised Indoor Localization. In *Proc. 10th ACM MobiSys*, pages 197–210, 2012.

[14] P. Wang, Z. Gao, X. Xu, Y. Zhou, H. Zhu, and K. Q. Zhu. Automatic Inference of Movements from Contact Histories. *SIGCOMM Computer Communication Review*, 41(4):386–387, 2011.

[15] C. Wu, Z. Yang, and Y. Liu. Smartphones Based Crowdsourcing for Indoor Localization. *IEEE Transactions on Mobile Computing (TMC)*, 14(2):444–457, 2015.

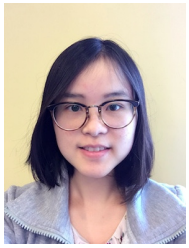
[16] H. Xie, T. Gu, X. Tao, H. Ye, and J. Lu. A Reliability-Augmented Particle Filter for Magnetic Fingerprinting based Indoor Localization on Smartphone. *IEEE Transactions on Mobile Computing (TMC)*, 15(8):1877–1892, 2016.

[17] J. Xiong and K. Jamieson. ArrayTrack: A Fine-Grained Indoor Location System. In *Proc. 10th USENIX NSDI*, pages 71–84, 2013.

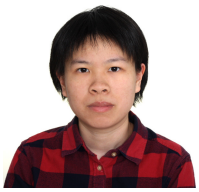
[18] M. Youssef and A. Agrawala. The Horus WLAN Location Determination System. In *Proc. of 3rd ACM MobiSys*, pages 205–218, 2005.

[19] F. Zafari and I. Papapanagiotou. Enhancing iBeacon based Micro-Location with Particle Filtering. In *IEEE GLOBECOM*, 2015.

[20] Y. Zheng, G. Shen, L. Li, C. Zhao, M. Li, and F. Zhao. Travi-navi: Self-deployable Indoor Navigation System. In *Proc. 20th ACM MobiCom*, pages 471–482, 2014.



Liyao Xiang (S) received the B.Eng. degree in Electrical and Computer Engineering from Shanghai Jiao Tong University, Shanghai, China in 2012 and the M.A.Sc. degree in Electrical and Computer Engineering from University of Toronto, Ontario, Canada in 2015. Her research interests include mobile computing, privacy and security in data mining.



Tzu Yin Tai received her Bachelor degree from the Division of Engineering Science in University of Toronto with a major in Electrical and Computer Engineering. She is currently a master student at Stanford University studying Computer Science, specializing in Computer Systems.

Baochun Li (F) received his B.E. degree from the Department of Computer Science and Technology, Tsinghua University, China, in 1995 and his M.S. and Ph.D. degrees from the Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, in 1997 and 2000.

Since 2000, he has been with the Department of Electrical and Computer Engineering at the University of Toronto, where he is currently a Professor. He holds the Nortel Networks Junior Chair in Network Architecture and Services from October 2003 to June 2005, and the Bell Canada Endowed Chair in Computer Engineering since August 2005. His research interests include cloud computing, large-scale data processing, computer networking, and distributed systems. In 2000, Dr. Li was the recipient of the IEEE Communications Society Leonard G. Abraham Award in the Field of Communications Systems. In 2009, he was a recipient of the Multimedia Communications Best Paper Award from the IEEE Communications Society, and a recipient of the University of Toronto McLean Award. He is a Fellow of IEEE and a member of ACM.

Bo Li (F) is a professor in the Department of Computer Science and Engineering, Hong Kong University of Science and Technology. He has been the Chief Technical Advisor for Chinacache Corp. (a NASDAQ listed company), the leading CDN operator in China since 2008. He held a Cheung Kong Visiting Chair Professor in Shanghai Jiao Tong University (2010-2013). He was an adjunct researcher at Microsoft Research Asia (1999-2007) and at Microsoft Advance Technology Center (2007-2009). His current research interests include data-center networking, cloud computing, content distribution in the Internet, and mobile wireless networking.

He made pioneering contributions in the Internet video broadcast with a system called Coolstreaming, which was credited as first large-scale Peer-to-Peer live video streaming system in the world. This work received the inaugural The Test-of-Time Paper Award from IEEE INFOCOM (2015). He has been an editor or a guest editor for over a two dozen of journals and magazines, mostly in IEEE and ACM. He was the Co-TPC Chair for IEEE INFOCOM 2004.

He received six Best Paper Awards from IEEE. He received the Young Investigator Award from Natural Science Foundation of China (NSFC) in 2005, the State Natural Science Award (2nd Class) in 2011. He received his PhD in the Electrical and Computer Engineering from University of Massachusetts at Amherst, and his B. Eng. (summa cum laude) in the Computer Science from Tsinghua University, Beijing, China.